Politechnika Wrocławska

Sztuczna inteligencja i inżynieria wiedzy

Laboratorium - Lista 5

20.06.2024

Krzysztof Głowacz, 266545

Spis treści

1 Wstęp

Laboratorium nr 5 dotyczyło wykorzystania sieci neuronowych w praktycznym problemie regresji. Jego celem było poznanie podstaw działania sieci neuronowej typu MLP (*Multi-layer Perceptron*) oraz wpływu poszczególnych jej hiperparametrów. Dalsze sekcje sprawozdania stanowią opis kolejnych kroków podjętych podczas rozwiązywania zadania. Obejmują one: podział zbioru danych na część uczącą i walidacyjną, sprawdzenie działania domyślnego modelu typu MLP o domyślnej konfiguracji, badanie wpływu tempa uczenia oraz rozmiaru modelu (liczby neuronów) na osiągane wyniki, a także przetestowanie ostatecznie wybranej konfiguracji na samodzielnie przygotowanych danych (żartach).

2 Przygotowanie danych

Przedmiotem analizy był zbiór Jester [2], który obejmował 100 różnych żartów wraz z ocenami ich śmieszności od niespełna 75 tysięcy osób. Żarty oceniane były w skali $(-10.0, 10.0)$. Przygotowanie danych obejmowało w pierwszej części połączenie 3 plików z ocenami (po ok. 24 tys. ocen na każdy z plików), a następnie dla każdego wiersza wybranie wszystkich kolumn poza pierwszą (gdyż ta zawierała informację o liczbie ocenionych żartów przez danego użytkownika) i zamianę wartości *99* na inną postać, która w dalszej analizie traktowana będzie jako *null*. Operacja ta była konieczna, ponieważ oryginalny zbiór danych właśnie liczbą *99* przedstawiał puste wartości (*nulle*). Ze względu na użycie biblioteki **numpy** w języku **Python** reprezentacją tą zostało wyrażenie: `NaN`. Ostatecznie zbiór wszystkich ocen miał rozmiar: 73421 wierszy $\times$ 100 kolumn i prezentował się następująco (pierwsze 10. wierszy, kolumny symbolizują numery kolejnych żartów):

	1	2	3	4	...	99	100
0	-7.82	8.79	-9.66	-8.16		NaN	NaN
1	4.08	-0.29	6.36	4.37		-4.32	1.07
2	NaN	NaN	NaN	NaN		NaN	NaN
3	NaN	8.35	NaN	NaN		NaN	NaN
4	8.50	4.61	-4.17	-5.39		1.80	1.60
5	-6.17	-3.54	0.44	-8.50		-5.05	-3.45
6	NaN	NaN	NaN	NaN		NaN	NaN
7	6.84	3.16	9.17	-6.21		1.31	0.00
8	-3.79	-3.54	-9.42	-6.89		-3.40	-4.95
9	3.01	5.15	5.15	3.01		NaN	NaN

Tabela 1: Przykładowe oceny śmieszności kolejnych żartów (od 10 osób)

Aby móc trenować model sieci neuronowej na zebranych danych konieczne było przypisanie jednej wartości dla każdego żartu, ponieważ celem modelu jest znalezienie takiej **funkcji**, która możliwie najlepiej opisze mapowanie treści żartu na poziom jego śmieszności. Funkcja natomiast z definicji jest takim przyporządkowaniem, które każdemu elementowi z jednego zbioru przyporządkowuje dokładnie jeden element z drugiego zbioru. Z tego powodu każdemu ze 100 żartów przypisana została średnia arytmetyczna z jego ocen. Otrzymano więc wektor długości 100, gdzie najmniejsza średnia ocen wyniosła: -3.705, a największa: 3.363. Wektor ten stał się wektorem etykiet `Y`.

W celu przygotowanie wektora tekstów `X` połączono sto plików w formacie *html*, z których każdy zawierał tekst danego żartu. Przy użyciu biblioteki **beautifulsoup4** pobrane zostały treści żartów, które po dodatkowym przetworzeniu (usunięciu znaków białych, usunięciu prefiksów „Q.” oraz „A.” dla żartów z pytaniem i odpowiedzią, itp.) trafiły do jednej listy. Finalnie pierwsze pięć elementów listy wyglądało następująco:

1. A man visits the doctor. The doctor says "I have bad news for you.You have cancer and Alzheimer's disease". The man replies "Well, thank God I don't have cancer!"
2. This couple had an excellent relationship going until one day he came home from work to find his girlfriend packing. He asked her why she was leaving him and she...
3. What's 200 feet long and has 4 teeth? The front row at a Willie Nelson Concert.

4. What's the difference between a man and a toilet? A toilet doesn't follow you around after you use it.

5. What's O. J. Simpson's Internet address? Slash, slash, backslash, slash, slash, escape.

Tak przygotowane teksty żartów należało następnie poddać procesowi wektoryzacji opartemu o tzw. słownik. Każda treść żartu przekształcona została na wektor $x \in \mathbb{N}^D$, który na i-tej pozycji zawierał liczbę wystąpień słowa/sylaby o indeksie i. Podczas laboratorium przetestowano dwa modele przetwarzania języka naturalnego: **Bert (bert-base-cased)** oraz **FastText**. W przypadku tego pierwszego długość wektora x równa była 768, a w przypadku drugiego 300. Tak przekształcone listy żartów stały się wektorami `X`.

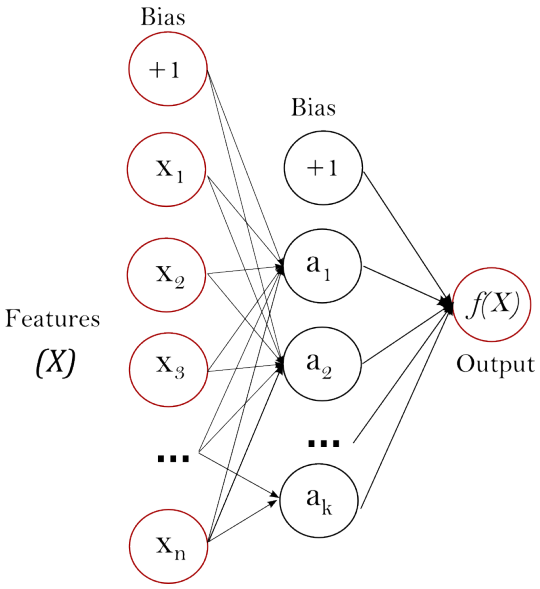
Ostatnim etapem przygotowania danych był podział wektorów `X` i `Y` na zbiory danych uczących i walidacyjnych. W tym celu skorzystano z funkcji `train_test_split` z biblioteki **sklearn.model_selection**. Ze względu na stosunkowo niewielki rozmiar zbioru danych wejściowych dokonano podziału w stosunku 3:7, tj. rozmiar zbioru danych testowych stanowił 30% rozmiaru zbioru wszystkich danych. Aby zapewnić powtarzalność całego procesu między kolejnymi wykonaniami wszystkie funkcje mogące zachowywać się w sposób pseudolosowy otrzymywały taki sam generator liczb pseudolosowych (z takim samym ziarnem).

3 Wykorzystanie modelu MLP

Multi-layer Perceptron (MLP) jest algorytmem uczenia nadzorowanego, który poszukuje funkcji

$$f : \mathbb{R}^m \to \mathbb{R}^o$$

(m - liczba wymiarów wejściowych, o - liczba wymiarów wyjściowych) poprzez trening na zbiorze danych uczących. Może zostać użyty w przypadku problemu regresji, a także klasyfikacji. Pomiędzy warstwami wejściową i wyjściową może znajdować się jedna lub więcej warstw, zwanych warstwami ukrytymi. Symboliczna reprezentacja sieci tego typu pochodząca z dokumentacji użytej biblioteki [1] zaprezentowana jest na poniższym zdjęciu:



Zdjęcie 1: Przykładowa sieć MLP z jedną warstwą ukrytą

W dalej opisanych eksperymentach używany był model `MLPRegressor`, którego najważniejsze hiperparametry (z punktu widzenia tego laboratorium) to:

1. **hidden_layer_sizes** - tablica, której i-ty element reprezentuje liczbę neuronów w i-tej warstwie ukrytej. We wszystkich testach używana była jedna warstwa ukryta. Domyślna wartość: *(100,)*.

2. **solver** - metoda optymalizacji wag. W testach używana była wartość *sgd*, która używa stochastyczny gradient do wyznaczenia kierunku poszukiwań. Domyślna wartość: *adam*.

3. **alpha** - siła regularyzacji L2. W testach używaną wartością było 0.0. Domyślna wartość: 0.0001.

4. **learning_rate** - strategia aktualizacji tempa uczenia. W testach ustawiona była wartość *constant*, która oznacza stałą prędkość uczenia parametryzowaną przez **learning_rate_init**. Domyślna wartość: *constant*.

5. **learning_rate_init** - początkowa prędkość uczenia, kontroluje wielkość „kroku" przy aktualizacji wag. Domyślna wartość: 0.001.

6. **random_state** - określa generator pseudolosowy używany przy niektórych wartościach, np. wagach. W testach używano `np.random.RandomState(42)` w celu zapewnienia powtarzalności między kolejnymi wykonaniami danego testu. Domyślna wartość: *None*.

7. **warm_start** - w przypadku ustawienia flagi `True` poprzednie rozwiązanie jest używane ponowne przy inicjalizacji następnego (w przeciwnym przypadku poprzednie rozwiązanie jest usuwane). W testach flaga była ustawiona na wartość `True`. Domyślna wartość: *False*.

8. **max_iter** - maksymalna liczba iteracji. W przypadku użycia metody *sgd* wartość ta określa liczbę epok. W testach ustawiona była wartość 1, ponieważ liczba epok kontrolowana była w inny sposób (przez zewnętrzny licznik), który umożliwiał łatwe tworzenie wykresów. Domyślna wartość: *200*.

3.1 Model MLP o domyślnej konfiguracji

Etap testowania wpływu poszczególnych hiperparametrów na jakość działania sieci neuronowej typu MLP rozpoczynał się od sprawdzenia działania podstawowego modelu o domyślnej konfiguracji. Kod odpowiedzialny za przygotowanie takiego podstawowego modelu wyglądał następująco:

```
rng = np.random.RandomState(42)

mlp = MLPRegressor(solver='sgd', alpha=0.0, learning_rate='constant', random_state=rng,
    warm_start=True, max_iter=1)
```

Wykresy na poniższych zdjęciach przedstawiają wartość funkcji kosztu (błąd średniokwadratowy) odpowiednio zbioru treningowego i walidacyjnego w zależności od epoki:

(a) Model **Bert**

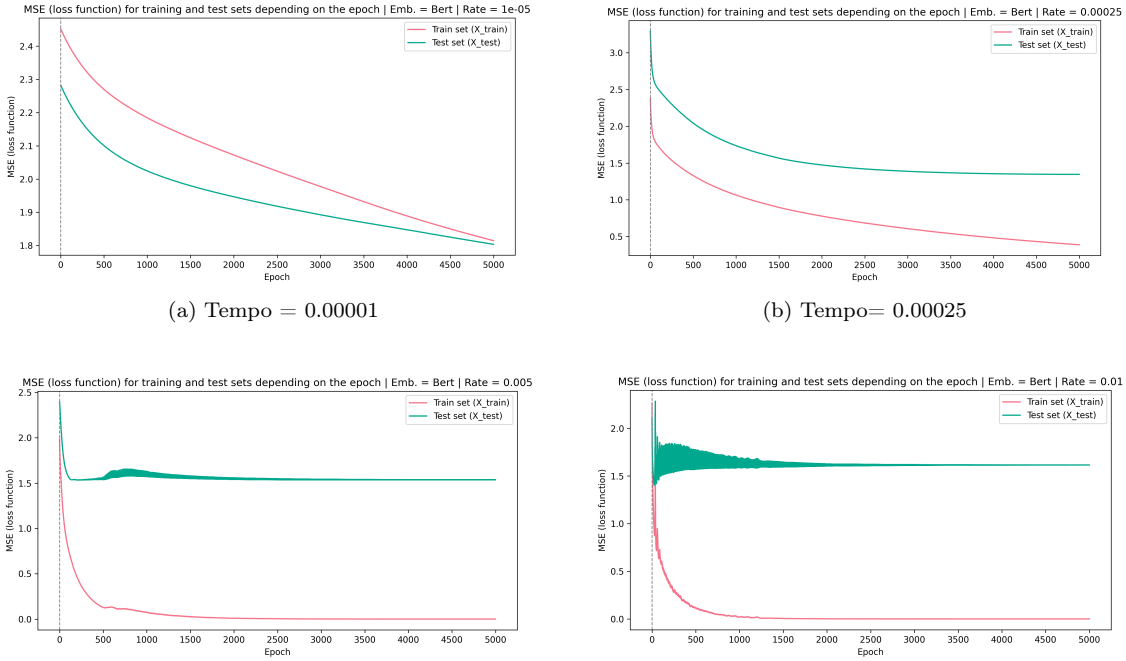
(b) Model **FastText**

Zdjęcie 2: Porównanie funkcji kosztu w zależności od epoki dla obu zbiorów, dla obu modeli NLP

Wykres dla modelu Bert świadczy o tym, że po ok. 500-1000 epokach pojawiło się zjawisko *overfittingu* - model MLP zaczął się zbyt dobrze dopasowywać do danych treningowych, o czym świadczą wartości MSE dla zbioru treningowego dążące do zera. Wartości MSE dla zbioru testowego natomiast przestały maleć, a zaczęły powoli rosnąć, tzn. model zaczął sobie gorzej radzić z predykowaniem wartości dla nieznanych próbek. Wykres dla modelu FastText z kolei nie wykazuje oznak *overfittingu*, ale generalnie widoczne jest bardzo małe dopasowanie do danych w przypadku obu zbiorów, a drobne różnice między wartościami MSE dla zbioru uczącego i walidacyjnego mogą wynikać z natury samych zbiorów - dane testowe mogą zawierać przykładowo więcej szumów i trudniejszych tekstów do predykowania.

3.2 Wpływ tempa uczenia na wyniki modelu

Następna część laboratorium i poznawania sieci typu MLP dotyczyła zbadania wpływu tempa uczenia (*learning rate*) na wyniki osiągane przez model. Wykonane zostały po 4 eksperymenty dla obu modeli NLP z wartościami tempa uczenia równymi kolejno: 0.00001, 0.00025, 0.005, 0.01 (domyślna wartość to 0.001). Wyniki przedstawiają wykresy ze zdjęcia 3.



(a) Tempo = 0.00001

(b) Tempo= 0.00025

(c) Tempo = 0.005

(d) Tempo = 0.01

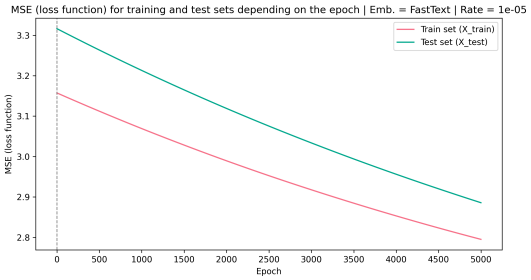
Zdjęcie 3: Porównanie funkcji kosztu w zależności od epoki i tempa uczenia dla obu zbiorów, model **Bert**

Zgodnie z oczekiwaniami przy najmniejszej wartości tempa uczenia, stukrotnie mniejszej od tej domyślnej, model nie wykazuje tak znaczących oznak *overfittingu*, ale wartości funkcji kosztu są istotnie większe od tych ze zdjęcia 2(a), ponieważ model uczy się w tym przypadku bardzo powoli i jeszcze nie zdążył się tak bardzo dopasować do danych. Dla tempa uczenia 0.00025 wykres jest najbardziej zbliżony do tego domyślnego, widoczny jest już trend, że model zaczyna się mocno dopasowywać do danych treningowych, ale jednocześnie nie występują jeszcze tak wyraźne oznaki *overfittingu*. Dla dwóch pozostałych wartości tempa uczenia model niezwykle szybko osiągnął w zasadzie maksymalne dopasowanie do danych treningowych i stałe wartości MSE dla zbioru testowego. W obu przypadkach tempo uczenia było zdecydowanie zbyt wysokie.

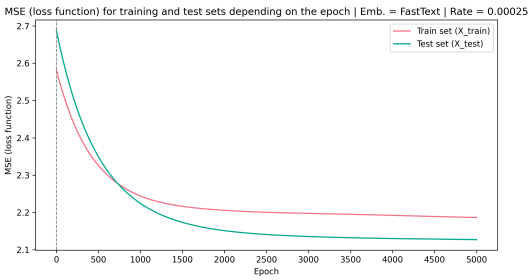
Analogiczny eksperyment został przeprowadzony dla modelu NLP FastText, a rezultaty ukazują wykresy ze zdjęcia 4. Na ich podstawie można stwierdzić, że w tym przypadku znacznie dłużej zajmowało modelowi dopasowanie się do danych. Oznaki tego zjawiska widoczne były dopiero dla dwóch największych wartości tempa uczenia, przy czym i tak w żadnej z nich wartości MSE dla zbioru treningowego nie zbliżyły się do poziomu 0. Generalnie w tym przypadku osiągane błędy średniokwadratowe dla zbioru testowego był wyższe niż w poprzednim, a model był mniej stabilny i przewidywalny. Przyczyną tego może być fakt, że podczas tokenizacji żartów model Bert zamienia tekst na wektor ponad 2.5 razy dłuższy niż wektor otrzymany z modelu FastText.

3.3 Wpływ liczby neuronów na wyniki modelu

Kolejna część eksperymentów przebiegała analogicznie - tym razem badany był wpływ rozmiaru sieci MLP na wyniki modelu. Wykonane zostały ponownie po 4 eksperymenty dla obu modeli NLP z liczbami neuronów równymi kolejno: 2, 50 i 200 (domyślna wartość to 100). Wyniki przedstawiają wykresy ze zdjęć 5 i 6.



(a) Tempo = 0.00001

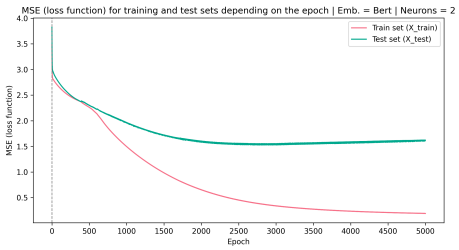


(b) Tempo= 0.00025

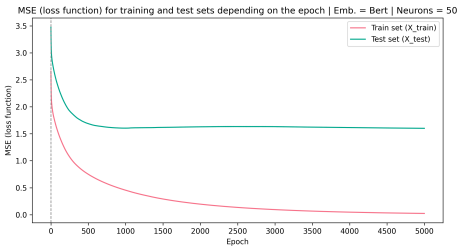
(c) Tempo = 0.005

(d) Tempo = 0.01

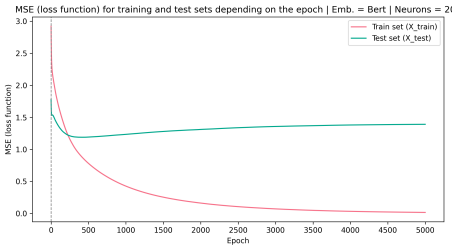
Zdjęcie 4: Porównanie funkcji kosztu w zależności od epoki i tempa uczenia dla obu zbiorów, model **FastText**



(a) Liczba neuronów = 2

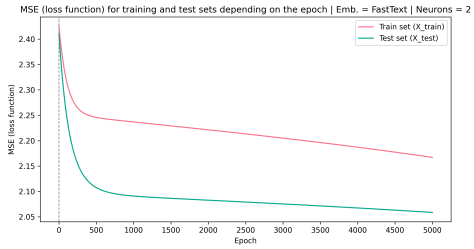


(b) Liczba neuronów = 50

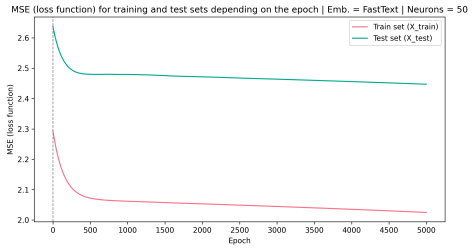


(c) Liczba neuronów = 200

Zdjęcie 5: Porównanie funkcji kosztu w zależności od epoki i liczby neuronów dla obu zbiorów, model **Bert**



(a) Liczba neuronów = 2



(b) Liczba neuronów = 50

(c) Liczba neuronów = 200

Zdjęcie 6: Porównanie funkcji kosztu w zależności od epoki i liczby neuronów dla obu zbiorów, model **FastText**

W tym przypadku można zauważyć, że modelu zupełnie nie był w stanie dopasować się do danych przygotowanych przy użyciu FastText niezależnie od podanej liczby neuronów. Może to wynikać z faktu, że domyślne tempo uczenia sieci (zastosowane w tej sytuacji) było zbyt wolne dla tych danych. W przypadku danych przygotowanych z użyciem Bert widzimy, że przy dwóch neuronach sieć nie była w stanie dopasować się do danych, dla 50 neuronów się to już udało, ale to w przypadku 200 neuronów, mimo wystąpienia zjawiska *overfittingu*, model najlepiej dopasował się do danych osiągając najniższe wartości MSE dla zbioru danych testowych ze wszystkich przeprowadzonych testów.

Literatura

[1] Dokumentacja biblioteki scikit-learn `https://scikit-learn.org/stable/modules/neural_networks_supervised.html`. Dostęp: 19.06.2024.

[2] Zbiór danych Jester `https://eigentaste.berkeley.edu/dataset/`. Dostęp: 19.06.2024.